JOURNAL OF COMPUTER AND SYSTEM SCIENCES 29, 80–98 (1984)

A Generalized Nexttime Operator in Temporal Logic

F. KRÖGER

Technical University Munich, Munich, West Germany

Received March 14, 1983; revised November 9, 1983

This paper introduces a new binary operator **atnext** into temporal logic generalizing the usual nexttime operator in a straightforward way. This operator has the same expressive power as the until operator and turns out to be a useful means for naturally describing and proving safety properties of programs.

1. INTRODUCTION

Temporal logic has proved itself a valuable tool for describing and verifying properties of programs (cf., for example, [8]). Extending usual first-order logic by some additional propositional operators, this logic allows for expressing the possible varying of assertions over *time*. The "classical" set of such operators—systematically introduced into the context of programs by Pnueli [10, 11] and Kröger [5, 6]—is given by

— the two *modal* operators $\Box$ ("always") and $\Diamond$ ("sometime") and

— the *nexttime* operator $\bigcirc$.

Application of these operators to an assertion A yields new assertions with the following informal meaning:

$\Box A$: "A will be true at every time in the future,"

$\Diamond A$: "A will be true at some time in the future,"

$\bigcirc A$: "A will be true at the next time point,"

where we assume some linear and discrete order of time.

In [1], Gabbay *et al.* argued that certain properties of programs cannot be expressed by these operators alone and suggested the additional use of

— the *until* operator **until**.

until is binary and applied to assertions A and B it means:

A **until** B: "B will be true at some time in the future and A will be true until then."

0022-0000/84 $3.00

until is very powerful in the sense that every other operator in a large class of temporal operators can be expressed by **until** together with the classical connectives $\neg$, $\wedge$, $\vee$, etc. [1, 4]. For example, we have

$$\Diamond A \leftrightarrow \textbf{true until } A,$$

$$\bigcirc A \leftrightarrow \textbf{false until } A.$$

Let us consider one of the examples for the use of **until** given in [1]. If A and B assert that some events a and b (resp.) are happening then

$$\neg B \textbf{ until } A$$

expresses the assertion that when b will happen next (if at all), a must have happened before.

When applying these linguistic means in practice to the description of program properties we made two observations:

— $\neg B$ **until** A actually asserts something more than noted above; it also says that a *will* eventually happen. But often it seems to be desirable to separate ("safety" or "invariance") properties as above from ("liveness" or "eventuality") properties as the latter one and not to pack them into one formula.

— Suppose the event a causes some "typical" predicate P_a to become true. In such a case we might be interested not so much in the assertion that a must happen before b is happening next time, but rather in an assertion like "when b will happen next, P_a will be true." (We will give an illustrating example in Section 3.)

In this paper, we want to investigate an operator **atnext** expressing just the latter phrase. In general, its meaning is:

A **atnext** B: "A will be true at the next time point that B is true"
(not assuming that A or B will be true at all).

We can see already at this informal stage that **atnext** is a rather natural generalization of the nexttime operator $\bigcirc$, modifying "the next time point" to "the next time point that B." $\bigcirc A$ is expressed as a special case by **atnext**:

$$\bigcirc A \leftrightarrow A \textbf{ atnext true}.$$

Since **false** can never be true we have furthermore that

$$\Box A \leftrightarrow \textbf{false atnext } \neg A.$$

This shows (together with $\Diamond A \leftrightarrow \neg\Box\neg A$) that $\Box$ and $\Diamond$ are also expressible by this new operator.

In the following sections of this paper we want to give

— some more formal treatment of **atnext** in the context of programs,

— examples of its use,

— some formal logical properties of **atnext**, in particular its comparison with **until**, and an axiomatization.

2. The Atnext Operator

We give now more formal definitions.

Let $\mathscr{L}$ be some usual first-order language with connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ augmented with the additional syntactic rule that

$$\bigcirc A,\ \Box A,\ \Diamond A,\ A \textbf{ until } B \text{ and } A \textbf{ atnext } B$$

are formulas if A and B are.

For notational simplicity we establish a priority order $\neg$, $\bigcirc$, $\Box$, $\Diamond$, **until**, **atnext**, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ of the operators with $\neg$ binding most and $\leftrightarrow$ binding least.

A *Kripke structure* $\mathbb{K}$ for $\mathscr{L}$ is given by

— a denumerable sequence $W = \{\eta_0, \eta_1, \eta_2, \ldots\}$ of *states*; η_0 is called the *initial state*,

— a mapping V associating a *truth value* $V(A, \eta_i) \in \{\mathbf{t}, \mathbf{f}\}$ with every atomic formula A of $\mathscr{L}$ and every $\eta_i \in W$. ($V(A, \eta_i)$ is the "truth value of A in state η_i.")

V is inductively extended to all formulas A of $\mathscr{L}$:

(1) $V(\neg A, \eta_i) = \mathbf{t}$ iff $V(A, \eta_i) = \mathbf{f}$,

(2) $V(A \wedge B, \eta_i) = \mathbf{t}$ iff $V(A, \eta_i) = \mathbf{t}$ and $V(B, \eta_i) = \mathbf{t}$,

and in the same way for $\vee, \rightarrow, \leftrightarrow$,

(3) $V(\bigcirc A, \eta_i) = \mathbf{t}$ iff $V(A, \eta_{i+1}) = \mathbf{t}$,

(4) $V(\Box A, \eta_i) = \mathbf{t}$ iff $V(A, \eta_j) = \mathbf{t}$ for all $j > i$,

(5) $V(\Diamond A, \eta_i) = \mathbf{t}$ iff $V(A, \eta_j) = \mathbf{t}$ for some $j > i$,

(6) $V(A \textbf{ until } B, \eta_i) = \mathbf{t}$ iff $V(B, \eta_j) = \mathbf{t}$ for some $j > i$ and $V(A, \eta_k) = \mathbf{t}$ for all k, $i < k < j$,

(7) $V(A \textbf{ atnext } B, \eta_i) = \mathbf{t}$ iff $V(B, \eta_j) = \mathbf{f}$ for all $j > i$ or $V(A, \eta_k) = \mathbf{t}$ for the smallest $k > i$ with $V(B, \eta_k) = \mathbf{t}$.

Note that whenever $V(B, \eta_k) = \mathbf{t}$ for some η_k, there is a (unique) smallest such k.

For some technical reasons we have chosen (as in [1]) the operators $\Box$ and $\Diamond$ not to include the *present* state in taking $j > i$ and not $j \geqslant i$ in (4) and (5). In many cases just the latter is desirable, but it can easily be defined via $\Box$ and $\Diamond$. Denoting the corresponding operators by $\dot{\Box}$ and $\dot{\Diamond}$ we take

$$\dot{\Box} A \equiv A \wedge \Box A,$$

$$\dot{\Diamond} A \equiv A \vee \Diamond A.$$

A formula A is called *valid in* $\mathbb{K}$ if $V(A, \eta_i) = \mathbf{t}$ for all $\eta_i \in W$. A is called *valid*, if A is valid in every Kripke structure $\mathbb{K}$.

We have already noted some valid formulas in Section 1, e.g.,

$$\Diamond A \leftrightarrow \neg\Box\neg A,$$

$$\bigcirc A \leftrightarrow A \textbf{ atnext true},$$

etc.

There are many formulas, in particular concerning $\bigcirc$, $\Box$ and $\Diamond$, known to be valid. We do not repeat them here (see, e.g., [2]). We rather want to note some valid formulas concerning the new operator:

(F1) A **atnext** A

(F2) A **atnext** $C \wedge B$ **atnext** $C \leftrightarrow (A \wedge B)$ **atnext** C

(F3) A **atnext** $C \vee B$ **atnext** $C \leftrightarrow (A \vee B)$ **atnext** C

(F4) $\Box(B \to A) \to A$ **atnext** B

(F5) $\Box(A \to B) \to (A$ **atnext** $C \to B$ **atnext** $C)$

(F6) $\Box\neg B \to A$ **atnext** B

(F7) $\bigcirc(A \wedge B) \to A$ **atnext** B

(F8) $\bigcirc(\neg B \wedge A$ **atnext** $B) \to A$ **atnext** B

(F9) $\dot{\Box}(A \wedge B \to A$ **atnext** $B) \to (A \wedge B \to \Box(B \to A))$

The validity of these formulas is quite obvious. Consider, e.g., (F8). Suppose a state η_i in which A **atnext** B is false. This means that there is a smallest $j > i$ with B true and A false in η_j. If $j = i + 1$ then $\neg B$ is false in η_{i+1}; if $j > i + 1$ then A **atnext** B is false in η_{i+1} because then j is also the smallest $k > i + 1$ with B true in η_k. Hence, in any case, $\bigcirc(\neg B \wedge A$ **atnext** $B)$ is false in η_i.

(F8) will be a vital part of the axiom system given in Section 5.

Of particular interest is the formula (F9) which generalizes the usual *induction principle* expressed by

$$\dot{\Box}(A \to \bigcirc A) \to (A \to \Box A).$$

We will prove (F9) axiomatically in Section 5. Informally, (F9) states that given $A \wedge B$ is true in some state η_i and $A \wedge B$ always implies that A is true when B is true next time then whenever B will be true after η_i, A will be true, too. We will come back to this in the next section.

Let now

$$\Pi \equiv \textbf{initial } P;$$

$$\textbf{cobegin } \Pi_1 \| \Pi_2 \| \cdots \| \Pi_n \textbf{ coend}$$

denote a (parallel) program where P is some initial condition and $\Pi_1, \ldots, \Pi_n$ are sequential programs (cf. Pnueli [12]). With every occurrence of an indivisible

statement in Π we associate a unique name (label). Let $\mathfrak{A}_\Pi$ be the set of all these names. Every $\alpha \in \mathfrak{A}_\Pi$ denotes an "action" of Π and so instead of a phrase "the action denoted by α" we simply read "the action α."

As a model of computation of Π we adopt the usual interleaving of all $\alpha \in \mathfrak{A}_\Pi$ respecting, of course, the sequential orderings given by $\Pi_1, \ldots, \Pi_n$.

In order to combine such a program Π with the language $\mathscr{L}$ from above we simply allow every $\alpha \in \mathfrak{A}_\Pi$ to be an atomic formula of $\mathscr{L}$, informally meaning the assertion that "α is executed." The interleaving model then means that

(Π1) In every state, exactly one $\alpha \in \mathfrak{A}_\Pi$ is true.

Furthermore, we introduce a special atomic formula start which is true only in the initial state. This allows for incorporating the initial condition P:

(Π2) start $\rightarrow P$ is true.

Note that since Π might be terminating, we assume a nil action running forever after Π in this case in order to "fill up" the infinite state sequence. Furthermore, our program description in $\mathscr{L}$ differs somewhat from the usual one (cf. [2, 12]). We come back to this in the next section.

A formula A, valid under the additional conditions (Π1) and (Π2), is obviously a "true" assertion about the program Π. We call A in this case *Π-valid*, denoted by $\Pi \Vdash A$. Of course, every valid formula is Π-valid.

Most of the formulas (F1)–(F9) listed above describe laws of how one can find valid formulas of the kind A **atnext** B. These laws are "purely logical." We can now ask for such rules in the context of some (arbitrary) program Π, i.e., rules for deriving Π-valid formulas.

Let us consider a quite simple but useful rule of this kind. Suppose that executing an action α in a state where A holds provides that B holds in the next state, as a formula,

$$\alpha \wedge A \rightarrow \bigcirc B,$$

and suppose furthermore that B is an invariant of all other actions of Π, then we can be sure that B will be true when α will be executed next time. So we have the following rule:

(R1) If $\Pi \Vdash \alpha \wedge A \rightarrow \bigcirc B$
and $\Pi \Vdash \beta \wedge B \rightarrow \bigcirc B$ for every $\beta \in \mathfrak{A}_\Pi$, $\beta \neq \alpha$
then $\Pi \Vdash \alpha \wedge A \rightarrow B$ **atnext** α.

(We now prefer to note such rules as "proof rules" rather than as implications like before.)

(R1) is a kind of *invariance rule*. Note that we do not (want to) conclude that B is true whenever α is executed but only the next time this happens. This enables us to think of many further rules of this kind. The general type of formulas inferred by these rules would be

$$\alpha \wedge A \rightarrow B \textbf{ atnext } \beta$$

(not necessarily $\beta = \alpha$). We believe indeed that a systematic and powerful set of various proof rules for such formulas meeting different "invariance behaviours" of properties would help a lot for practical comprehension of larger verification problems.

Being far from having such a systematic view, let us only note a further example to illustrate what we mean:

(R2) If in some Π_i, $1 \leqslant i \leqslant n$, β immediately follows α
and $\Pi \Vdash \alpha \wedge A \to \bigcirc B$
and $\Pi \Vdash \gamma \wedge B \to \bigcirc B$ for all γ in all Π_j, $j \neq i$
then $\Pi \Vdash \alpha \wedge A \to B$ **atnext** β.

We may also think of even more general goals. We may want to prove that B is true next time β is executed and some additional condition C holds, i.e.,

$$\alpha \wedge A \to B \textbf{ atnext } (\beta \wedge C).$$

Again as an example, we give a rule for $\alpha = \beta$:

(R3) If $\Pi \Vdash \alpha \wedge A \to \bigcirc B$
and $\Pi \Vdash \beta \wedge B \to \bigcirc B$ for every $\beta \in \mathfrak{A}_\Pi$, $\beta \neq \alpha$
and $\Pi \Vdash \alpha \wedge B \wedge \neg C \to \bigcirc B$
then $\Pi \Vdash \alpha \wedge A \to B$ **atnext** $(\alpha \wedge C)$.

(R3) is a straightforward generalization of (R1).

3. Applications of the Atnext Operator

We now want to give an example of how **atnext** could profitably be used.

Suppose $U = u_0, u_1, u_2, \ldots$ to be an infinite sequence of "messages" and let U be the input for the following program:

$\Pi \equiv$ **initial** nextinput $= u_0 \wedge nr = 1 \wedge ls = mn = a = 0$;
cobegin
sender: **loop** ρ_a: **if** $ls = a$ **then** $ls := ls \oplus 1$;
$d :=$ nextinput **fi**;
σ_m: send (ls, d) to (mn, inf)
endloop
$\|$receiver: **loop** ρ_m: **if** $mn = nr$ **then** nextoutput := inf;
$nr := nr \oplus 1$ **fi**;
σ_a: send $(nr \oplus 1)$ to (a)
endloop
coend

Π describes the essence of the *alternating bit protocol.* The send operations in σ_m and σ_a are thought to be carried out over an "unreliable medium" which might corrupt messages.

ρ_a, σ_m, ρ_m, σ_a are the elements of $\mathfrak{A}_\Pi$ with the following informal meanings:

ρ_a: receive acknowledgment,
σ_m: send message (thought of as writing the compound message (ls, d) into (mn, inf)),
ρ_m: receive message,
σ_a: send acknowledgment (writing $nr \oplus 1$ into a).

A more formal specification will be given below. $\oplus$ denotes addition modulo 2.

For a more detailed discussion of the protocol we refer the reader to, e.g., [2, 3, 13]. It should be noted, however, that our program is really parallel whereas the comparable programs in [2, 3] are essentially sequential since there the media are though as one-element buffers so that send and receive operations of messages and acknowledgments are forced to run in a strongly alternating way. In Π, we allow overwriting of messages in the medium, so a message may be corrupted again after being sent correctly and before taken from the respective receiver.

Nevertheless, Π guarantees to transmit all messages of U in the same order to an output sequence (provided some additional conditions on fairness and the functioning of the units).

This fact can be expressed by the following "correctness" formulas:

(C1) $\text{start} \to \text{inf} = u_0$ **atnext** $(\rho_m \wedge mn = nr)$

(C2) $\rho_m \wedge mn = nr \wedge \text{inf} = u_i \to \text{inf} = u_{i+1}$ **atnext** $(\rho_m \wedge mn = nr)$

(C3) $\Diamond(\rho_m \wedge mn = nr)$.

Since $\rho_m \wedge mn = nr$ means "Π is outputting," (C1) and (C2) describe the claim that if there is output at all then it is the desired one. (C3) tells us that in every state there really will be some next output.

Let us make some remarks about this correctness specification:

(i) We find it convenient that the pure liveness claim (C3) is separated from the safety assertions (C1) and (C2) as already noted in the Introduction. So one can prove it once for the two applications concerning u_0 and u_{i+1}.

(ii) Remembering our motivation of the atnext operator in Section 1 we could have written, e.g.,

$$\neg(\rho_m \wedge mn = nr) \textbf{ until } (\rho_m \wedge mn = nr \wedge \text{inf} = u_{i+1})$$

instead of

$$\text{inf} = u_{i+1} \textbf{ atnext } (\rho_m \wedge mn = nr)$$

in (C2). Besides the separation effect already mentioned the atnext formulation (and its proof) appears simpler and closer to the intuitive usage of language. Note, by the way, that

$$\neg(\rho_m \wedge mn = nr) \textbf{ until } \text{inf} = u_{i+1}$$

would obviously not be sufficient.

(iii) Instead of our atomic formulas $\alpha \in \mathfrak{A}_\Pi$, another kind of such formulas, mostly denoted by $at\alpha$, is used frequently (e.g., [2, 12]). $at\alpha$ informally means "α is ready to execute" (but there may be executed some β from another parallel component first). There may be some other events between $at\alpha$ and α and so we could not simply take $at\rho_m$ in this sense instead of ρ_m in (C1)–(C3). This is the reason for introducing the formulas α instead of $at\alpha$.

Next we want to comment on how to prove (C1)–(C3). The liveness (C3) can be proved relatively simply provided some assumptions about the fairness and about the eventual correct functioning of the transmission media of the system. We do not want to go into the details here because it is not necessary to use the atnext operator anywhere (although it is even possible). We rather want to give an outline of a possible proof of (C2).

For this purpose we have first to specify the effect of the actions of Π somewhat more precisely:

(1) $\rho_a \wedge ls = a \wedge (\text{nextinput}, ls) = (u_i, ls_0)$
$\rightarrow \bigcirc[(\text{nextinput}, ls, d) = (u_{i+1}, ls_0 \oplus 1, u_i)$ and all other variables remain unchanged$]$

(2) $\rho_a \wedge ls \neq a \wedge A \rightarrow \bigcirc A$ for arbitrary A

(3) $\sigma_m \rightarrow \bigcirc[(mn, \text{inf}) = (ls, d) \vee (mn, \text{inf}) = (\text{error}, \text{error})$,
all other variables remain unchanged$]$

(4) $\rho_m \wedge mn = nr = nr_0 \rightarrow \bigcirc[nr = nr_0 \oplus 1$ and all other variables (except nextoutput) remain unchanged$]$

(5) $\rho_m \wedge mn \neq nr \wedge A \rightarrow \bigcirc A$ for arbitrary A

(6) $\sigma_a \rightarrow \bigcirc[a = nr \oplus 1 \vee a = \text{error}$,
all other variables remain unchanged$]$

These "specifications" are only partially formal but we want to leave the discussion on this level. Completely formal specifications could be carried out, for example, in a similar way as in [6]. It should be noted that using the nexttime operator in these specifications, both safety *and* liveness properties of the atomic statements are expressed (compare [9]).

Secondly we note our initial condition:

(7) $\text{start} \rightarrow \text{nextinput} = u_0 \wedge nr = 1 \wedge ls = mn = a = 0.$

Proof of (C2). We first note some Π-valid formulas of the form $\dot{\Box}A$. Every one of these formulas can be very easily proved by showing that $\text{start} \rightarrow A$ is Π-valid and A is an invariant of every $\alpha \in \mathfrak{A}_\Pi$, i.e., $\alpha \wedge A \rightarrow \bigcirc A$ is Π-valid:

(8) $\dot{\Box}(nr \in \{0, 1\} \wedge ls \in \{0, 1\})$

(9) $\dot{\Box}[(mn = nr \to nr = ls \wedge \inf = d) \wedge (a = ls \to nr \neq ls)]$

(10) $\dot{\Box}(d = u_i \to \text{nextinput} = u_{i+1})$.

From (9) we infer that

$$\rho_m \wedge mn = nr \wedge \inf = u_i \to \rho_m \wedge mn = nr \wedge nr = ls \wedge ls \neq a \wedge d = u_i,$$

and with (4) we get

(11) $\rho_m \wedge mn = nr \wedge \inf = u_i \to \bigcirc(mn \neq nr \wedge nr \neq ls \wedge d = u_i)$.

Informally, (11) means that immediately after outputting u_i, we have $mn \neq nr \wedge nr \neq ls \wedge d = u_i$. Our goal is to show that $\inf = u_{i+1}$ next time ρ_m is carried out with $mn = nr$ being true. According to rule (R3) noted in Section 2 we try to find a "local" invariant B such that

— $mn \neq nr \wedge nr \neq ls \wedge d = u_i$ implies B,

— B is an invariant of ρ_a, σ_m, σ_a and of ρ_m (if $mn \neq nr$),

— B **atnext** $(\rho_m \wedge mn = nr)$ implies $\inf = u_{i+1}$ **atnext** $(\rho_m \wedge mn = nr)$.

The simplest approach to B is to look at $mn \neq nr \wedge nr \neq ls \wedge d = u_i$, or better $mn \neq nr \wedge nr \neq ls \wedge \text{nextinput} = u_{i+1}$ (which is implied using (10)) itself. This formula can easily be seen to be invariant under σ_m, σ_a and ρ_m (if $mn \neq nr$) but executing ρ_a may lead to a state where $nr = ls \wedge d = u_{i+1}$ holds. So next we try to take

$$B = (mn \neq nr \wedge nr \neq ls \wedge \text{nextinput} = u_{i+1}) \vee (nr = ls \wedge d = u_{i+1})$$

and, in fact, this B is an appropriate choice. With (10) we have

$$mn \neq nr \wedge nr \neq ls \wedge d = u_i \to B$$

and therefore

(12) $\rho_m \wedge mn = nr \wedge \inf = u_i \to \bigcirc B$.

B can easily be checked to be an invariant of ρ_a, σ_m and σ_a:

(13) $\alpha \wedge B \to \bigcirc B$ for $\alpha \in \{\rho_a, \sigma_m, \sigma_a\}$.

Furthermore we have with (5)

(14) $\rho_m \wedge mn \neq nr \wedge B \to \bigcirc B$.

Applying rule (R3) to (12), (13), (14) we get

(15) $\rho_m \wedge mn = nr \wedge \inf = u_i \to B$ **atnext** $(\rho_m \wedge mn = nr)$.

With

$$B \textbf{ atnext } (\rho_m \wedge mn = nr) \rightarrow (B \wedge mn = nr) \textbf{ atnext } (\rho_m \wedge mn = nr)$$

which is an application of (F1) and (F2) and using (9) once more we finally get

$$\rho_m \wedge mn = nr \wedge \inf = u_i \rightarrow \inf = u_{i+1} \textbf{ atnext } (\rho_m \wedge mn = nr)$$

which was to be proved.

This proof is a good illustration of how to use the atnext operator. The proof of (C1) runs very similarly. All together, we get a quite transparent proof of this nontrivial program. Note, by the way, that it appears as a simpler proof of a more complicated program than in [2, 3], in particular a proof without any additional technical expense caused, e.g., by history variables.

We want to conclude this section by briefly indicating that the atnext operator is not only a tool for formulating safety properties in a natural way but also for proving usual safety properties of the form

$$A \rightarrow \Box B.$$

Let us illustrate this by a trivial example (we take a sequential program for simplicity)

$$\Pi \equiv \textbf{initial } \mathrm{B};$$
$$\textbf{loop } \alpha_1; \alpha_2; \ldots; \alpha_m \textbf{ endloop}.$$

Suppose we want to prove that some Q is a loop invariant, i.e., Q holds whenever execution is at α_1. The essential part of this claim is to show

(I1) $\alpha_1 \wedge Q \rightarrow \Box(\alpha_1 \rightarrow Q)$.

Informally, (I1) is true if "going once around" the loop, Q is true again provided it was true before, i.e., if

$$\begin{aligned} (16)\quad & \alpha_1 \wedge Q \rightarrow \bigcirc(\alpha_2 \wedge Q_2) \\ & \alpha_2 \wedge Q_2 \rightarrow \bigcirc(\alpha_3 \wedge Q_3) \\ & \vdots \\ & \alpha_m \wedge Q_m \rightarrow \bigcirc(\alpha_1 \wedge Q). \end{aligned}$$

Suppose (16) has been proved. A usual formal proof of (I1) then runs as follows: Let $Q_1 \equiv Q$ and

$$P \equiv [(\alpha_1 \rightarrow Q_1) \wedge (\alpha_2 \rightarrow Q_2) \wedge \cdots \wedge (\alpha_m \rightarrow Q_m)].$$

Then we have

(17) $\alpha_i \wedge Q_i \to P$ for every $i = 1,..., m$

(18) $\alpha_i \wedge P \to Q_i$ for every $i = 1,..., m$.

From (16), (17) and (18) we find that P is an invariant of every α_i, hence

(19) $P \to \bigcirc P$.

By the usual induction principle (cf. Section 2), we get

(20) $P \to \Box P$,

hence with (17), (18), (20) we get (I1).

The same idea can now be presented in a different way: Applying the formula (F7) of Section 2 to the last implication of (16) and then successively applying (F8) to (16) (from below), we find

(21) $\alpha_m \wedge Q_m \to Q$ **atnext** α_1

$\alpha_{m-1} \wedge Q_{m-1} \to Q$ **atnext** α_1

$\vdots$

$\alpha_1 \wedge Q \to Q$ **atnext** α_1.

Applying (F9) to the last implication of (21) we directly get (I1).

This shows that (F9) generalizes the usual induction principle in a very elegant way. Besides notational convenience and brevity the use of the atnext operator in such proofs seems again to be very close to informal reasoning. As before, this extends to more general applications as, for example, to proofs of assertions like

(I2) $\alpha_i \wedge Q \wedge R \to \Box(\alpha_i \wedge R \to Q)$.

In (I2), Q is a *generalized invariant* which need not hold everytime at α_i but only when R holds at α_i (compare (C2) in the previous example). In such cases even the proof idea may differ from a usual invariant proof (cf. [7]).

4. Formal Properties of the Atnext Operator

After having illustrated how to apply the new operator we want to conclude the consideration of **atnext** by showing some formal (and purely logical) properties, in particular a comparison of **atnext** and **until**.

First we want to remember four obviously valid equivalences already mentioned in Section 1:

(F10) $\bigcirc A \leftrightarrow$ **false until** A

(F11) $\bigcirc A \leftrightarrow A$ **atnext true**

(F12) $\Diamond A \leftrightarrow$ **true until** A

(F13) $\Box A \leftrightarrow$ **false atnext** $\neg A$.

Of course, because of the duality of $\Diamond$ and $\Box$, we also can express $\Box$ by **until** and $\Diamond$ by **atnext**:

(F14) $\Box A \leftrightarrow \neg($**true until** $\neg A)$

(F15) $\Diamond A \leftrightarrow \neg($**false atnext** $A)$.

Next we want to show how the atnext operator can be expressed by **until**. We have already seen in Section 3 that A **atnext** B can "almost" be expressed by $\neg B$ **until** $(A \wedge B)$ with the only difference that in A **atnext** B, B could also never hold. This observation leads to

(F16) A **atnext** $B \leftrightarrow \neg B$ **until** $(A \wedge B) \vee \Box\neg B$.

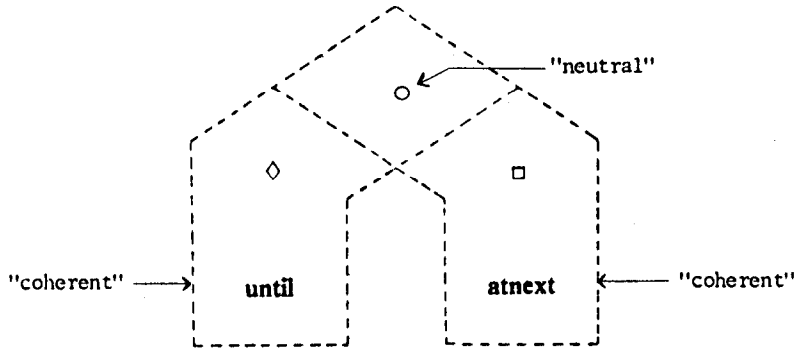
Proof. A **atnext** B being true in some state η_i means that either B is false for η_j, $j > i$, i.e., $\Box\neg B$ is true in η_i, or there is a smallest $k > i$ with B true in η_k and A is true in η_k, too. $\neg B$ **until** $(A \wedge B)$ being true in η_i means just the same: the fact that $j > i$ exists with $A \wedge B$ true in η_j is equivalent to the fact that there exists a smallest j of this kind and this j is the k of above.

The expressibility of **atnext** by **until** could be expected because of the expressive power of **until**. More interesting is the other direction which shows that **atnext** has the same power:

(F17) A **until** $B \leftrightarrow B$ **atnext** $(A \rightarrow B) \wedge \Diamond B$.

Proof. A **until** B being true in η_i means that there exists $j > i$ and, equivalently, even a smallest $j > i$ such that B is true in η_j and A is true in η_k, $i < k < j$. The first part means that $\Diamond B$ is true in η_i. The second part means the same as B **atnext** $(A \rightarrow B)$ being true in η_i: Because of the minimal choice of j, B is false in η_k for $i < k < j$, hence j is also the smallest j such that $A \rightarrow B$ is true in η_j.

Both operators **until** and **atnext** can express eventuality and invariance assertions. However, if we remember the use of **atnext** in Section 3 (describing safety properties, proving invariances), if, furthermore, we compare (F13) and (F15), we have the "feeling" that, from its nature, **atnext** "corresponds" or "harmonizes" much more with $\Box$ than with $\Diamond$. **until**, on the other hand, does not contain this close coherency with $\Box$ (e.g., there is no natural connection between **until** and $\Box$ like (F9)), it rather matches with $\Diamond$ (compare (F12) and (F14)). The operator $\bigcirc$ being trivial special applications of both **until** and **atnext** is "neutral" with respect to this classification. Summarizing this view of the operators we may draw the following picture.



In this picture, furthermore, we know $\Diamond$ and $\Box$ to be dual to each other in the sense that

$$\neg\Diamond A \leftrightarrow \Box\neg A$$

and

$$\neg\Box A \leftrightarrow \Diamond\neg A.$$

We may ask whether there is some similar connection between **until** and **atnext**. Let us first note the following two equivalences:

(F18) $\neg(A \textbf{ atnext } B) \leftrightarrow \neg A \textbf{ atnext } B \wedge \Diamond B$

(F19) $\neg(A \textbf{ until } B) \leftrightarrow \neg B \textbf{ until } (\neg A \wedge \neg B) \vee \Box\neg B.$

Proof. A **atnext** B being false in state η_i means that B is true in some η_j, $j > i$, and A will not be true in η_j with minimal such j. This is just what $\Diamond B$ and $\neg A$ **atnext** B means when being true in η_i. The proof of (F19) is left to the reader.

(F18) and (F19) show how to express negated atnext or until assertions by unnegated ones (plus $\Diamond$ or $\Box$, resp.). Now we show how negations of **atnext** and **until** can be expressed by (unnegated) **until** and **atnext**, resp., by noting the following

Duality principle of **until** and **atnext**:

(F20) $\neg(A \textbf{ atnext } B) \leftrightarrow \neg B \textbf{ until } (\neg A \wedge B)$

(F21) $\neg(A \textbf{ until } B) \leftrightarrow \neg B \textbf{ atnext } (\neg A \vee B).$

Proof. Applying (F18) and (F16), we find

$$\begin{aligned}\neg(A \textbf{ atnext } B) &\leftrightarrow \neg A \textbf{ atnext } B \wedge \Diamond B\\ &\leftrightarrow (\neg B \textbf{ until } (\neg A \wedge B) \vee \Box\neg B) \wedge \Diamond B\\ &\leftrightarrow \neg B \textbf{ until } (\neg A \wedge B) \wedge \Diamond B\\ &\leftrightarrow \neg B \textbf{ until } \quad (\neg A \wedge B).\end{aligned}$$

For the proof of (F21), we first note that $\Box C \rightarrow C$ **atnext** D is valid and hence C **atnext** $D \vee \Box C \leftrightarrow C$ **atnext** D holds. Furthermore, $\Diamond(C \vee D) \vee \neg\Diamond D$ is always true, so we have with (F17) and (F18):

$$\neg(A \textbf{ until } B) \leftrightarrow \neg(B \textbf{ atnext } (A \rightarrow B)) \vee \neg\Diamond B$$
$$\leftrightarrow (\neg B \textbf{ atnext } (\neg A \vee B) \wedge \Diamond(\neg A \vee B)) \vee \neg\Diamond B$$
$$\leftrightarrow (\neg B \textbf{ atnext } (\neg A \vee B) \vee \Box\neg B) \wedge (\Diamond(\neg A \vee B) \vee \neg\Diamond B)$$
$$\leftrightarrow \neg B \textbf{ atnext } (\neg A \vee B).$$

The duality described in (F20) and (F21) is not fully analogous to what is usually meant by "dual." Full duality in this sense would be the equivalence of $\neg(A \textbf{ atnext } B)$ and $\neg B \textbf{ until } \neg A$ and in the same way for **until**. Such equivalences do not hold but since

$$\neg B \textbf{ until } (\neg A \wedge B) \rightarrow \neg B \textbf{ until } \neg A$$

and

$$\neg B \textbf{ atnext } (\neg A \vee B) \rightarrow \neg B \textbf{ atnext } \neg A$$

can very easily be recognized to be valid, we deduce from (F20) and (F21) the additional

Weak duality principle of **until** and **atnext**:

(F22) $\neg(A \textbf{ atnext } B) \rightarrow \neg B \textbf{ until } \neg A$

(F23) $\neg(A \textbf{ until } B) \rightarrow \neg B \textbf{ atnext } \neg A$.

5. An Axiomatization of the Atnext Operator

So far most of our argumentations about logical properties of **atnext** have been semantical. We now provide an axiom system Σ and indicate by some examples how proofs can be carried out axiomatically. Σ formalizes a (propositional part of) temporal logic containing the operators $\bigcirc$, $\Box$ and **atnext** (taking $\Diamond$ and **until** as "defined" operators introduced by (F15) and (F17)).

Axioms:

(Σ1) All instances of tautologies of usual propositional logic

(Σ2) $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

(Σ3) $\bigcirc\neg A \leftrightarrow \neg\bigcirc A$

(Σ4) $\bigcirc(A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B)$

(Σ5) $\Box A \rightarrow \bigcirc A \wedge \bigcirc\Box A$

(Σ6) $\Box(A \rightarrow \bigcirc A) \rightarrow (\bigcirc A \rightarrow \Box A)$

(Σ7) $\Box\neg B \rightarrow A \textbf{ atnext } B$

(Σ8) $A \textbf{ atnext } B \leftrightarrow \bigcirc(A \wedge B) \vee \bigcirc(\neg B \wedge A \textbf{ atnext } B)$

Rules of inference:

($\Sigma 9$) $A, A \to B \vdash B$

($\Sigma 10$) $A \vdash \Box A$

This formal system is very similar to a system DUX given in [1]. We have only replaced the two axioms

(U1) A **until** $B \to \Diamond B$

(U2) A **until** $B \leftrightarrow \bigcirc B \vee (\bigcirc A \wedge \bigcirc(A$ **until** $B))$

of DUX by the axioms ($\Sigma 7$) and ($\Sigma 8$) for the atnext operator.

The axioms and rules of Σ are easily shown to be sound with respect to the semantics given in Section 2. Furthermore, according to (F17), (U1) is the same as

$$B \textbf{ atnext } (A \to B) \wedge \Diamond B \to \Diamond B$$

and this formula is an axiom ($\Sigma 1$), hence derivable in Σ. In the same way (U2) is

(U2′) B **atnext** $(A \to B) \wedge \Diamond B \leftrightarrow \bigcirc B \vee (\bigcirc A \wedge \bigcirc(B$ **atnext** $(A \to B) \wedge \Diamond B))$.

We will prove below that this can be reduced to

(U2″) B **atnext** $(A \to B) \wedge \Diamond B \leftrightarrow [\bigcirc((A \to B) \wedge B)$
$\vee \bigcirc(\neg(A \to B) \wedge B$ **atnext** $(A \to B))] \wedge \Diamond B$

which is again derivable in Σ since

$$B \textbf{ atnext } (A \to B) \leftrightarrow \bigcirc((A \to B) \wedge B) \vee \bigcirc(\neg(A \to B) \wedge B \textbf{ atnext } (A \to B))$$

is an instance of axiom ($\Sigma 8$).

These arguments show that Σ is also a complete system since DUX is complete.

We now want to give some sample formal derivations within Σ. In order to shorten the proofs we first note four quite trivial derived rules:

(prop) If B "follows propositionally" from $A_1, \ldots, A_n$ (i.e., $A_1 \wedge \cdots \wedge A_n \to B$ is an axiom ($\Sigma 1$)), then $A_1, \ldots, A_n \vdash B$.

A typical example of (prop) is the "cut" or "chain reasoning" rule

$$A \to B, B \to C \vdash A \to C$$

which we will frequently use in the following.

(nex) $A \vdash \bigcirc A$

(mon 1) $A \to B \vdash \Box A \to \Box B$

(mon 2) $A \to B \vdash \bigcirc A \to \bigcirc B$

(prop) follows immediately using ($\Sigma 1$) and ($\Sigma 9$). (nex) is derived as follows:

(1)	A	assumption
(2)	$\Box A$	($\Sigma 10$), (1)
(3)	$\Box A \to \bigcirc A \wedge \bigcirc\Box A$	($\Sigma 5$)
(4)	$\Box A \to \bigcirc A$	(prop), (3)
(5)	$\bigcirc A$	($\Sigma 9$), (2), (4)

Derivation of the "monotonicity" rule (mon 1):

(1)	$A \to B$	assumption
(2)	$\Box(A \to B)$	($\Sigma 10$), (1)
(3)	$\Box(A \to B) \to (\Box A \to \Box B)$	($\Sigma 2$)
(4)	$\Box A \to \Box B$	($\Sigma 9$), (2), (3)

(mon 2) is derived in the same way using (nex) instead of ($\Sigma 10$).

We now give a derivation of the formula

$$(\bigcirc A \to \bigcirc B) \to \bigcirc(A \to B).$$

(1)	$\neg(A \to B) \to A$	($\Sigma 1$)
(2)	$\bigcirc(\neg(A \to B) \to A)$	(nex), (1)
(3)	$\bigcirc\neg(A \to B) \to \bigcirc A$	($\Sigma 4$), ($\Sigma 10$), (2)
(4)	$\neg\bigcirc(A \to B) \to \bigcirc\neg(A \to B)$	($\Sigma 3$), (prop)
(5)	$\neg\bigcirc(A \to B) \to \bigcirc A$	(prop), (3), (4)
(6)	$\neg(A \to B) \to \neg B$	($\Sigma 1$)
(7)	$\neg\bigcirc(A \to B) \to \bigcirc\neg B$	like (5)
(8)	$\bigcirc\neg B \to \neg\bigcirc B$	($\Sigma 3$), (prop)
(9)	$\neg\bigcirc(A \to B) \to \neg\bigcirc B$	(prop), (7), (9)
(10)	$\neg\bigcirc(A \to B) \to \bigcirc A \wedge \neg\bigcirc B$	(prop), (5), (9)
(11)	$(\bigcirc A \to \bigcirc B) \to \bigcirc(A \to B)$	(prop), (10)

This formula together with ($\Sigma 4$) means that $\bigcirc(A \to B) \leftrightarrow (\bigcirc A \to \bigcirc B)$ is valid and together with ($\Sigma 3$) one finds that $\bigcirc$ "distributes" over every classical operator $\neg$, $\to$, $\wedge$, $\vee$, etc. We will use this principle also as a derived rule and indicate it by (dist).

We now want to complete the argument made before in showing the completeness of Σ. Applying (prop) and (dist) we find

$$\bigcirc B \vee (\bigcirc A \wedge \bigcirc(B \textbf{ atnext } (A \to B) \wedge \Diamond B))$$

$$\leftrightarrow [\bigcirc B \vee (\bigcirc A \wedge \bigcirc(B \textbf{ atnext } (A \to B)))] \wedge (\bigcirc B \vee \bigcirc\Diamond B)$$

$$\leftrightarrow [\bigcirc((A \to B) \wedge B) \vee \bigcirc(\neg(A \to B) \wedge B \textbf{ atnext } (A \to B))] \wedge (\bigcirc B \vee \bigcirc\Diamond B)$$

and applying ($\Sigma 8$) we have

$$\begin{aligned} \bigcirc B \vee \bigcirc\Diamond B &\leftrightarrow \bigcirc B \vee \bigcirc\neg(\textbf{false atnext } B) \\ &\leftrightarrow \bigcirc(\textbf{true} \vee \neg B) \wedge \bigcirc(B \vee \neg(\textbf{false atnext } B)) \\ &\leftrightarrow \neg(\textbf{false atnext } B) \\ &\leftrightarrow \Diamond B \end{aligned}$$

which then shows the equivalence of (U2′) and (U2″).

We conclude this section by giving formal proofs for the basic relationships (F11) and (F13) and the generalized induction principle (F9).

Derivation of (F11): $\bigcirc A \leftrightarrow A$ **atnext true**

(1)	A **atnext true** $\leftrightarrow \bigcirc((A \wedge \textbf{true}) \vee (\neg\textbf{true} \wedge A \textbf{ atnext true}))$	($\Sigma 8$), (dist)
(2)	$(A \wedge \textbf{true}) \vee (\neg\textbf{true} \wedge A \textbf{ atnext true}) \leftrightarrow A$	($\Sigma 1$)
(3)	$\bigcirc A \leftrightarrow A$ **atnext true**	(prop), (mon 2), (1), (2)

Derivation of (F13): $\Box A \leftrightarrow$ **false atnext** $\neg A$

(1)	$\Box A \rightarrow \Box\neg\neg A$	($\Sigma 1$), (mon1)
(2)	$\Box A \rightarrow$ **false atnext** $\neg A$	($\Sigma 7$), (prop), (1)
(3)	**false atnext** $\neg A$ $\rightarrow \bigcirc[(\textbf{false} \wedge \neg A) \vee (\neg\neg A \wedge \textbf{false atnext } \neg A)]$	($\Sigma 8$), (dist), (prop)
(4)	**false atnext** $\neg A \rightarrow \bigcirc(A \wedge$ **false atnext** $\neg A)$	($\Sigma 1$), (mon2), (3)
(5)	$A \wedge$ **false atnext** $\neg A \rightarrow \bigcirc(A \wedge$ **false atnext** $\neg A)$	(prop), (4)
(6)	$\Box(A \wedge$ **false atnext** $\neg A \rightarrow \bigcirc(A \wedge$ **false atnext** $\neg A))$	($\Sigma 10$), (5)
(7)	$\bigcirc(A \wedge$ **false atnext** $\neg A) \rightarrow \Box(A \wedge$ **false atnext** $\neg A)$	($\Sigma 6$), ($\Sigma 9$), (6)
(8)	**false atnext** $\neg A \rightarrow \Box(A \wedge$ **false atnext** $\neg A)$	(prop), (4), (7)
(9)	**false atnext** $\neg A \rightarrow \Box A$	($\Sigma 1$), (mon1), (8)
(10)	$\Box A \leftrightarrow$ **false atnext** $\neg A$	(prop), (2), (9)

In order to derive (F9) we first prove the following.

Deduction theorem for Σ: If $A \vdash B$, then $\vdash \overset{\cdot}{\Box} A \rightarrow B$.

The proof of this theorem runs by induction on the derivation of B from A. If $B \equiv A$ or B is an axiom of Σ then clearly $\vdash \overset{\cdot}{\Box} A \rightarrow B$.

If B is reached by rule ($\Sigma 9$) from premises C and $C \rightarrow B$, then by induction hypothesis, $\vdash \overset{\cdot}{\Box} A \rightarrow C$ and $\vdash \overset{\cdot}{\Box} A \rightarrow (C \rightarrow B)$ from which we get $\vdash \overset{\cdot}{\Box} A \rightarrow B$ by (prop). If, finally, $B \equiv \Box C$ is reached by ($\Sigma 10$) from C and we can assume $\vdash \overset{\cdot}{\Box} A \rightarrow C$ then we derive $\overset{\cdot}{\Box} A \rightarrow \Box C$ in the following way:

(1)	$\dot{\Box}A \to C$	assumption
(2)	$\Box\dot{\Box}A \to \Box C$	(Σ10), (Σ2), (Σ9), (1)
(3)	$\dot{\Box}A \to \bigcirc\dot{\Box}A$	with (Σ5)
(4)	$\bigcirc\dot{\Box}A \to \Box\dot{\Box}A$	with (Σ6)
(5)	$\dot{\Box}A \to \Box C$	(prop), (2), (3), (4)

Derivation of (F9): $\dot{\Box}(A \wedge B \to A \textbf{ atnext } B) \to (A \wedge B \to \Box(B \to A))$.
Because of the deduction theorem it suffices to derive

$$A \wedge B \to A \textbf{ atnext } B \vdash A \wedge B \to \Box(B \to A).$$

(1)	$A \wedge B \to A \textbf{ atnext } B$	assumption
(2)	$A \textbf{ atnext } B \to \bigcirc((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B))$	(Σ8), (dist), (prop)
(3)	$A \wedge B \to \bigcirc((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B))$	(prop), (1), (2)
(4)	$\neg B \wedge A \textbf{ atnext } B \to \bigcirc((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B))$	(prop), (2)
(5)	$(A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B) \to \bigcirc((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B))$	(prop), (3), (4)
(6)	$\bigcirc((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B)) \to \Box((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B))$	(Σ6), (Σ9), (Σ10), (5)
(7)	$A \wedge B \to \Box((A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B))$	(prop), (3), (6)
(8)	$(A \wedge B) \vee (\neg B \wedge A \textbf{ atnext } B) \to (B \to A)$	(Σ1)
(9)	$A \wedge B \to \Box(B \to A)$	(mon1), (prop), (7), (8)

6. Concluding Remarks

One of the appealing features of temporal logic seems to be the fact that—as long as the operators $\bigcirc$, $\Diamond$ and $\Box$ are concerned—the formal logical language meets very nicely some intermediate, non-formalized level of describing program behaviours and reasoning about them (like first-order predicate logic meets the "normal" mathematical language). This seems to be not so clear with the until operator.

The atnext operator introduced in this paper has again this desirable property. It is a very natural descriptional tool and it has transparent and informally usable laws and proof rules as well for itself as in connection with $\bigcirc$ and $\Box$.

This does not mean, of course, that we want to replace all until assertions by atnext assertions (although this is possible as we have seen). But at least in the particular field of safety properties, **atnext** seems to be a more appropriate operator than **until**.

REFERENCES

1. D. GABBAY, A. PNUELI, S. SHELAH, AND J. STAVI, On the temporal analysis of fairness, *in* "Proceedings, Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, January 1980," pp. 163–173.
2. B. T. HAILPERN, Verifying concurrent processes using temporal logic, *in* "Lecture Notes in Computer Science No. 129," Springer-Verlag, New York/Berlin, 1982.
3. B. T. HAILPERN AND S. S. OWICKI, Modular verification of computer communication protocols, *IEEE Trans. Comm.* **Com-31** (1983), 56–67.
4. H. W. KAMP, "Tense Logic and the Theory of Linear Order," Ph.D. thesis, University of California, Los Angeles, 1968.
5. F. KRÖGER, Logical rules of natural reasoning about programs, *in* "Proceedings 3rd International Symposium on Automata, Languages and Programming, Edinburgh, England, July 1976," pp. 87–98.
6. F. KRÖGER, A uniform logical basis for the description, specification and verification of programs, *in* "Proceedings, IFIP Working Conference on Formal Description of Programming Concepts, St. Andrews, Canada, August 1977," pp. 441–457, North-Holland, Amsterdam, 1978.
7. F. KRÖGER, Infinite proof rule for loops, *Acta Inform.* **14** (1980), 371–389.
8. Z. MANNA, Logics of programs, *in* "Information Processing 80," pp. 41–51, North-Holland, Amsterdam, 1980.
9. S. S. OWICKI AND L. LAMPORT, Proving liveness properties of concurrent programs, *ACM Trans. Program. Lang. Syst.* **4** (1982), 455–495.
10. A. PNUELI, The temporal logic of programs, *in* "Proceedings, 18th Annual Symposium on Foundations of Computer Science, Providence, R.I., November 1977," pp. 46–57.
11. A. PNUELI, The temporal semantics of concurrent computation, *in* "Proceedings, Symposium on Semantics of Concurrent Computation, Evian, France, July 1979," pp. 1–20, Springer Pub., New York, 1979.
12. A. PNUELI, The temporal semantics of concurrent programs, *Theoret. Comput. Sci.* **13** (1981), 45–60.
13. R. L. SCHWARTZ AND P. M. MELLIAR-SMITH, From state machines to temporal logic: Specification methods for protocol standards, *IEEE Trans. Comm.* **Com-30** (1982), 2486–2496.